Logical relations for coherence of effect subtyping

Dariusz Biernacki and Piotr Polesiuk

Institute of Computer Science, University of Wrocław
Joliot-Curie 15, 50-383 Wrocław, Poland
{dabi,ppolesiuk}@cs.uni.wroc.pl

Abstract

A coercion semantics of a programming language with subtyping is typically defined on typing derivations rather than on typing judgments. To avoid semantic ambiguity, such a semantics is expected to be coherent, i.e., independent of the typing derivation for a given typing judgment. In this article we present heterogeneous, biorthogonal, step-indexed logical relations for establishing the coherence of coercion semantics of programming languages with subtyping. To illustrate the effectiveness of the proof method, we develop a proof of coherence of a type-directed, selective CPS translation from a typed call-by-value lambda calculus with delimited continuations and control-effect subtyping. The article is accompanied by a Coq formalization that relies on a novel shallow embedding of a logic for reasoning about step-indexing.

1998 ACM Subject Classification D.3.3 Language Constructs and Features, F.3.3 Studies of Program Constructs

Keywords and phrases type system, coherence of subtyping, logical relation, control effect, continuation-passing style

Digital Object Identifier 10.4230/LIPIcs.TLCA.2015.x

1 Introduction

Programming languages that allow for subtyping, i.e., a mechanism facilitating coercions of expressions of one type to another, are usually given either a subset semantics, where one type is considered a subset of another type, or – a coercion semantics, where expressions are explicitly converted from one type to another. In the presence of subtyping, typing derivations depend on the occurrences of the subtyping judgments and, therefore, typing judgments do not have unique typing derivations. Consequently, a coercion semantics that interprets subtyping judgment by introducing explicit type coercions is defined on typing derivations rather than on typing judgments. But then a natural question arises as to whether such a semantics is coherent, i.e., whether it does not depend on the typing derivation.

The problem of coherence has been considered in a variety of typed lambda calculi. Reynolds proved the coherence of the denotational semantics for intersection types [22]. Breazu-Tannen et al. proved the coherence of a coercion translation from the lambda calculus with polymorphic, recursive and sum types to system F [8], by showing that any two derivations of the same judgment are normalizable to a unique normal derivation where the correctness of the normalization steps is justified by an equational theory in the target calculus. Curien and Ghelli introduced a translation from system $F_{\leq}$ to a calculus with explicit coercions and showed that any two derivations of the same judgment are translated to terms that are normalizable to a unique normal form [9]. Finally, Schwinghammer followed Breazu-Tannen et al.'s approach to prove the coherence of coercion translation from Moggi's computational lambda calculus with subtyping [24].

The normalization-based proofs consist in finding a normal form for a representation of the derivation and they hinge on showing that such normal forms are unique for a given

13th International Conference on Typed Lambda Calculi and Applications (TLCA'15).
Editor: Thorsten Altenkirch; pp. 42–57

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

typing judgment. When the source calculus under consideration is presented in the spirit of the lambda calculus à la Church, i.e., the lambda abstractions are type annotated, as is the case in all the aforementioned articles that follow the normalization-based approach, the term and the typing context indeed determine the shape of the normal derivation (modulo a top level coercion that depends on the type of the term) [18]. However, in calculi à la Curry this is no longer the case and the method cannot be directly applied. Still, if the calculus is at least weakly normalizing, one can hope to recover the uniqueness property for normal typing derivations for source terms in normal form, assuming that term normalization preserves the coercion semantics. For instance, in the simply typed λ-calculus the typing context uniquely determines the type of the term in the function position in applications building a β-normal form, and, hence, derivations in normal form for such terms are unique. This line of reasoning cannot be used when the calculus includes recursion.

In this article, we consider the coherence problem in simply-typed lambda calculi with general recursion, control effects and with no type annotations. The coercion semantics we study translate typing derivations in the source calculus to a corresponding target calculus with explicit type coercions (that in most cases can be further replaced with equivalent lambda-term representations) and our criterion for coherence of the translation is contextual equivalence [19] in the target calculus.

The main result of this work is a construction of logical relations for establishing so construed coherence of coercion semantics, applicable in a variety of calculi. In particular, we address the problem of coherence of a type-directed CPS translation from the call-by-value λ-calculus with delimited-control operators and control-effect subtyping introduced by Materzok and the first author [16], extended with recursion. While the translation for the calculus with explicit type annotations has been shown to be coherent in terms of an equational theory in a target calculus [15], no CPS coercion translation for the original version, let alone extended with recursion, has been proven coherent.

The reasons why coherence in this calculus is important are twofold. First of all, it is very expressive and therefore interesting from the theoretical point of view. In particular, the calculus has been shown to generalize the canonical type-and-effect system for Danvy and Filinski's shift and reset control operators [10, 11], and, furthermore, that it is strictly more expressive than the CPS hierarchy of Danvy and Filinski [17]. These results heavily rely on the effect subtyping relation that, e.g., allows to coerce pure expressions to effectful ones. From a more practical point of view, the selective CPS translation, that leaves pure (i.e., control-effect free) expressions in direct style and introduces explicit coercions to interpret effect subtyping in the source calculus, is a good candidate for embedding the control operators in an existing programming language, such as Scala [23].

In order to deal with the complexity of the source calculus and of the translation itself, we introduce binary logical relations on terms of the target calculus that are: heterogeneous, biorthogonal [14, 20, 13], and step-indexed [3, 2, 1]. Heterogeneity allows us to relate terms of different types, and in particular those in continuation-passing style with those in direct style. This is a crucial property, since the same term can have a pure type, resulting in a direct-style term through the translation and another, impure type, resulting in a term in continuation-passing style. Relating such terms requires quantification over types and to assure well-foundedness of the construction, we need to use step-indexing, which also supports reasoning about recursion, even if not in a critical way. We follow Dreyer et al. [12] in using logical step-indexed logical relations in our presentation of step-indexing. Biorthogonality, by imposing a particular order of evaluation on expressions, simplifies the construction of the logical relations. It also facilitates reasoning about continuations represented as evaluation

contexts.

Apart from the calculus with effect subtyping, we have used the ideas presented in this article to show coherence of subtyping in several other calculi, including the simply typed lambda calculus with subtyping [18] extended with recursion, the calculus of intersection types [22], and the lambda calculus with subtyping and the control operator call/cc.

The article is accompanied by a Coq development that presently consists of a library that provides a new shallow embedding of the logic for reasoning about step-indexed logical relations, and complete formalization of the proofs presented in the rest of the article. The code is available at `http://www.ii.uni.wroc.pl/~ppolesiuk/lrcoherence`.

The rest of this article is structured as follows. In Section 2, we briefly present Dreyer et al.'s logic for reasoning about step indexing [12] on which we base our presentation. We also describe the main ideas behind our Coq formalization of the logic. In Section 3, we introduce the construction of the logical relations in a simple yet sufficiently interesting scenario – the simply typed lambda calculus à la Curry with natural numbers, type Top, general recursion and standard subtyping. The goal of this section is to introduce the basic ingredients of the proof method before embarking on a considerably more challenging journey in the subsequent section. In Section 4, we present the main result of the article – the logical relations for establishing the coherence of the CPS translation from the calculus of delimited control with effect subtyping. In Section 5, we summarize the article.

2 Reasoning about step-indexed logical relations

Step-indexed logical relations [3, 2, 1] are a powerful tool for reasoning about programming languages. Instead of describing a general behavior of program execution, they focus on the first n computation steps, where the step index n is an additional parameter of the relation. This additional parameter makes it possible to define logical relations inductively not only on the structure of types, but also on the number of computation steps that are allowed for a program to make and, therefore, they provide an elegant way to reason about features that introduce non-termination to the programming language, including recursive types [2] and references [1].

However, reasoning directly about step-indexed logical relations is tedious because proofs become obscured by step-index arithmetic. Dreyer et al. [12] proposed logical step-indexed logical relations (LSLR) to avoid this problem. The LSLR logic is an intuitionistic logic for reasoning about one particular Kripke model: where possible worlds are natural numbers (step-indices) and where future worlds have smaller indices than the present one. All formulas are interpreted as monotone (non-increasing) sequences of truth values, whereas the connectives are interpreted as usual. In particular, in the case of implication we quantify over all future worlds to ensure monotonicity, so the formula $\varphi \Rightarrow \psi$ is valid at index n (written $n \models \varphi \Rightarrow \psi$) iff $k \models \varphi$ implies $k \models \psi$ for every $k \leq n$. In contrast to Dreyer et al. we do not assume that all formulas are valid in world 0, because it is not necessary.

The LSLR logic is also equipped with a modal operator $\triangleright$ (later), to provide access to strictly future worlds. The formula $\triangleright\varphi$ means φ *holds in any future world*, or formally $\triangleright\varphi$ is always valid at world 0, and $n+1 \models \triangleright\varphi$ iff φ is valid at n (and other future worlds by monotonicity). The later operator comes with two inference rules:

$$\frac{\Gamma, \Sigma \vdash \varphi}{\Gamma, \triangleright\Sigma \vdash \triangleright\varphi}\ \triangleright\text{-intro} \qquad \frac{\Gamma, \triangleright\varphi \vdash \varphi}{\Gamma \vdash \varphi}\ \text{Löb}$$

The first rule allows one to shift reasoning to a future world, making the assumptions about the future world available. The Löb rule expresses an induction principle for indices. Note

$$\begin{array}{lcll} \tau & ::= & \mathsf{Nat} \mid \mathsf{Top} \mid \tau \to \tau & \text{(types)} \\ e & ::= & x \mid \lambda x.e \mid e\ e \mid \mathsf{fix}\ x(x).e \mid n & \text{(expressions)} \end{array}$$

$$\frac{}{\tau \le \tau}\ \text{S-Refl} \qquad \frac{\tau_2 \le \tau_3 \qquad \tau_1 \le \tau_2}{\tau_1 \le \tau_3}\ \text{S-Trans} \qquad \frac{}{\tau \le \mathsf{Top}}\ \text{S-Top}$$

$$\frac{\tau_2' \le \tau_1' \qquad \tau_1 \le \tau_2}{(\tau_1' \to \tau_1) \le (\tau_2' \to \tau_2)}\ \text{S-Arr} \qquad \frac{(x:\tau) \in \Gamma}{\Gamma \vdash x\ :\ \tau}\ \text{T-Var} \qquad \frac{\Gamma, x:\tau_1 \vdash e\ :\ \tau_2}{\Gamma \vdash \lambda x.e\ :\ \tau_1 \to \tau_2}\ \text{T-Abs}$$

$$\frac{\Gamma \vdash e_1\ :\ \tau_2 \to \tau_1 \qquad \Gamma \vdash e_2\ :\ \tau_2}{\Gamma \vdash e_1\ e_2\ :\ \tau_1}\ \text{T-App} \qquad \frac{\Gamma, f:\tau_1 \to \tau_2, x:\tau_1 \vdash e\ :\ \tau_2}{\Gamma \vdash \mathsf{fix}\ f(x).e\ :\ \tau_1 \to \tau_2}\ \text{T-Fix}$$

$$\frac{}{\Gamma \vdash n\ :\ \mathsf{Nat}}\ \text{T-Const} \qquad \frac{\Gamma \vdash e\ :\ \tau \qquad \tau \le \tau'}{\Gamma \vdash e\ :\ \tau'}\ \text{T-Sub}$$

Figure 1 The source language – λ-calculus with subtyping

that the premise of the rule also captures the base case, because the assumption $\triangleright\varphi$ is trivial in the world 0. The later operator comes with no general elimination rule.

Predicates in LSLR logic as well as step-indexed logical relations can be defined inductively on indices. More generally, we can define a recursive predicate $\mu r.\varphi(r)$, provided all occurrences of r in φ are guarded by the later operator, to guarantee well-foundedness of the definition. For the sake of readability, in this paper we define recursive predicates and relations by giving a set of clauses instead of using the μ operator.

Since the logic is developed for reasoning about one particular model, we can freely add new inference rules for the logic if we prove they are valid in the model. We can also add new relations or predicates to the logic if we provide their monotone interpretation. In particular, constant functions are monotone, so we can safely use predicates defined outside of the logic, such as typing or reduction relations.

Our Coq formalization accompanying this article is built on our IxFree library that contains a shallow embedding of the LSLR logic similar to Appel et al.'s formalization of the "very modal model" [4]. Logical connectives including the later operator are functions on a special type `IProp` of "indexed propositions" defined as a type of monotone functions from `nat` to `Prop`. The library provides tactics representing the most important inference rules. One of the main differences between our library and Appel et al.'s formalization is a way of keeping track of the assumptions. Instead of interpreting a sequent $\varphi_1, \ldots, \varphi_n \vdash \psi$ directly, we treat it as $k \models \psi$ with the standard Coq assumptions $k \models \varphi_1$, $\ldots$, $k \models \varphi_n$. This approach is very convenient since it allows for reusing a number of existing Coq tactics.

3 Introducing the logical relations

In this section we prove the coherence of subtyping in the simply-typed call-by-value lambda calculus extended with recursion, where the coercion semantics is given by a standard translation to the simply-typed lambda calculus with explicit coercions [9]. Our goal here is to introduce the proof method in a simple scenario, so that in Section 4 we can focus on issues specific to control effects. The logical relations we present in this section are biorthogonal and step-indexed, which is not strictly necessary but it makes the development more elegant. Furthermore, biorthogonality and step-indexing become crucial in handling more complicated calculi such as the one of Section 4 and, therefore, are essential for the method to scale.

$$
\begin{array}{lcll}
\tau & ::= & \mathsf{Nat} \mid \mathsf{Unit} \mid \tau \to \tau & \text{(types)} \\
c & ::= & \mathsf{id} \mid c \circ c \mid \mathsf{top} \mid c \to c & \text{(coercions)} \\
e & ::= & x \mid \lambda x.e \mid e\ e \mid c\ e \mid \mathsf{fix}\ x(x).e \mid n \mid \langle\rangle & \text{(expressions)} \\
v & ::= & x \mid \lambda x.e \mid \mathsf{fix}\ x(x).e \mid (c \to c)\ v \mid n \mid \langle\rangle & \text{(values)} \\
E & ::= & \square \mid E\ e \mid v\ E \mid c\ E & \text{(evaluation contexts)}
\end{array}
$$

$$
\frac{}{\mathsf{id} :: \tau \rhd \tau}\ \textsc{S-Refl} \qquad \frac{c_1 :: \tau_2 \rhd \tau_3 \qquad c_2 :: \tau_1 \rhd \tau_2}{c_1 \circ c_2 :: \tau_1 \rhd \tau_3}\ \textsc{S-Trans} \qquad \frac{}{\mathsf{top} :: \tau \rhd \mathsf{Unit}}\ \textsc{S-Top}
$$

$$
\frac{c_1 :: \tau_2' \rhd \tau_1' \qquad c_2 :: \tau_1 \rhd \tau_2}{c_1 \to c_2 :: (\tau_1' \to \tau_1) \rhd (\tau_2' \to \tau_2)}\ \textsc{S-Arr} \qquad \frac{}{\Gamma \vdash n\ :\ \mathsf{Nat}}\ \textsc{T-Const} \qquad \frac{(x : \tau) \in \Gamma}{\Gamma \vdash x\ :\ \tau}\ \textsc{T-Var}
$$

$$
\frac{}{\Gamma \vdash \langle\rangle\ :\ \mathsf{Unit}}\ \textsc{T-Unit} \qquad \frac{\Gamma, x : \tau_1 \vdash e\ :\ \tau_2}{\Gamma \vdash \lambda x.e\ :\ \tau_1 \to \tau_2}\ \textsc{T-Abs} \qquad \frac{c :: \tau \rhd \tau' \qquad \Gamma \vdash e\ :\ \tau}{\Gamma \vdash c\ e\ :\ \tau'}\ \textsc{T-CApp}
$$

$$
\frac{\Gamma \vdash e_1\ :\ \tau_2 \to \tau_1 \qquad \Gamma \vdash e_2\ :\ \tau_2}{\Gamma \vdash e_1\ e_2\ :\ \tau_1}\ \textsc{T-App} \qquad \frac{\Gamma, f : \tau_1 \to \tau_2, x : \tau_1 \vdash e\ :\ \tau_2}{\Gamma \vdash \mathsf{fix}\ f(x).e\ :\ \tau_1 \to \tau_2}\ \textsc{T-Fix}
$$

$$
\begin{array}{rcl}
E[(\lambda x.e)\ v] & \to_\beta & E[e\{v/x\}] \\
E[(\mathsf{fix}\ f(x).e)\ v] & \to_\beta & E[e\{\mathsf{fix}\ f(x).e/f, v/x\}]
\end{array}
\qquad
\begin{array}{rcl}
E[\mathsf{id}\ v] & \to_\iota & E[v] \\
E[(c_1 \circ c_2)\ v] & \to_\iota & E[c_1\ (c_2\ v)] \\
E[\mathsf{top}\ v] & \to_\iota & E[\langle\rangle] \\
E[(c_1 \to c_2)\ v_1\ v_2] & \to_\iota & E[c_2\ (v_1\ (c_1\ v_2))]
\end{array}
$$

Figure 2 The target language – λ-calculus with explicit coercions

3.1 The simply-typed lambda calculus with subtyping

The syntax and typing rules for the source language are given in Figure 1. The language is the simply-typed lambda calculus with recursive functions ($\mathsf{fix}\ f(x).e$) and natural numbers (n). For brevity we do not consider any primitive operations on natural numbers or other basic types, but they could seamlessly be added to the language. We include the type Top, to make the subtyping relation interesting. The typing and subtyping rules are standard [18].

3.2 Coercion semantics

The semantics of the source language is given by a translation of the typing derivations to a target language that extends the source language with explicit type coercions (and replaces Top with Unit). The coercions express conversion of a term from one type to another, according to the subtyping relation. Figure 2 contains syntax, typing rules and reduction rules of the target language. The type coercions c and their typing rules correspond exactly to the subtyping rules of the source language. The grammar of terms contains explicit coercion application of the form $c\ e$ and it is worth noting that terms of the form $(c \to c)\ v$ are considered values, since they represent a coercion expecting another value as argument (witness the last reduction rule).

The operational semantics of the target language distinguishes between β-rules that perform actual computations and ι-rules that rearrange coercions. Both of them are used during program evaluation. We say that program e terminates (written $e{\downarrow}$) when it can be reduced to a value using both sorts of reduction rules, according to the evaluation strategy determined by the evaluation contexts.

General contexts are closed terms with one hole (possibly under some binders), and are denoted by the metavariable C. We write $\vdash C\ :\ (\Gamma; \tau_1)\ \rightsquigarrow\ \tau_2$ if for any e with $\Gamma \vdash e\ :\ \tau_1$ we have $\Gamma \vdash C[e]\ :\ \tau_2$. Contextual approximation, written $\Gamma \vdash e_1\ \precsim_{ctx}\ e_2\ :\ \tau$, means that

$$[\![\tau_1 \to \tau_2]\!] \;=\; [\![\tau_1]\!] \to [\![\tau_2]\!] \qquad [\![\mathsf{Nat}]\!] \;=\; \mathsf{Nat} \qquad [\![\mathsf{Top}]\!] \;=\; \mathsf{Unit}$$

$$\begin{aligned}
\mathcal{S}[\![\tau \le \tau]\!]_{\text{S-Refl}} &= \mathsf{id} \\
\mathcal{S}[\![\mathsf{Top} \le \tau]\!]_{\text{S-Top}} &= \mathsf{top} \\
\mathcal{S}[\![\tau_1 \le \tau_3]\!]_{\text{S-Trans}(D_1,D_2)} &= \mathcal{S}[\![\tau_2 \le \tau_3]\!]_{D_1} \circ \mathcal{S}[\![\tau_1 \le \tau_2]\!]_{D_2} \\
\mathcal{S}[\![\tau_1' \to \tau_1 \le \tau_2' \to \tau_2]\!]_{\text{S-Arr}(D_1,D_2)} &= \mathcal{S}[\![\tau_2' \le \tau_1']\!]_{D_1} \to \mathcal{S}[\![\tau_1 \le \tau_2]\!]_{D_2}
\end{aligned}$$

$$\begin{aligned}
\mathcal{T}[\![x]\!]_{\text{T-Var}} &= x & \mathcal{T}[\![\mathsf{fix}\, f(x).e]\!]_{\text{T-Fix}(D)} &= \mathsf{fix}\, f(x).\mathcal{T}[\![e]\!]_D \\
\mathcal{T}[\![\lambda x.e]\!]_{\text{T-Abs}(D)} &= \lambda x.\mathcal{T}[\![e]\!]_D & \mathcal{T}[\![e]\!]_{\text{T-Sub}(D_1,D_2)} &= \mathcal{S}[\![\tau \le \tau']\!]_{D_2}\, \mathcal{T}[\![e]\!]_{D_1} \\
\mathcal{T}[\![e_1\; e_2]\!]_{\text{T-App}(D_1,D_2)} &= \mathcal{T}[\![e_1]\!]_{D_1}\, \mathcal{T}[\![e_2]\!]_{D_2} & \mathcal{T}[\![n]\!]_{\text{T-Const}} &= n
\end{aligned}$$

Figure 3 Coercion semantics for the λ-calculus with subtyping

for any context C and type τ', such that $\vdash C \;:\; (\Gamma;\tau) \rightsquigarrow \tau'$ if $C[e_1]$ terminates, then so does $C[e_2]$. If $\Gamma \vdash e_1 \precsim_{ctx} e_2 \;:\; \tau$ and $\Gamma \vdash e_2 \precsim_{ctx} e_1 \;:\; \tau$, then we say that e_1 and e_2 are contextually equivalent.

The coercion semantics of the source language is given in Figure 3. The function $\mathcal{S}[\![.]\!]$ translates subtyping proofs into coercions, and function $\mathcal{T}[\![.]\!]$ translates typing derivations into terms of the target language.

▶ **Lemma 1.** *Coercion semantics preserves types.*
1. *If* $D :: \tau_1 \le \tau_2$ *then* $\mathcal{S}[\![\tau_1 \le \tau_2]\!]_D :: [\![\tau_1]\!] \triangleright [\![\tau_2]\!]$.
2. *If* $D :: \Gamma \vdash e \;:\; \tau$ *then* $[\![\Gamma]\!] \vdash \mathcal{T}[\![e]\!]_D \;:\; [\![\tau]\!]$.

3.3 Logical relations

In order to reason about contextual equivalence in the target language, we define logical relations (Figure 4). Relations are expressed in the LSLR logic described in Section 2, so they are implicitly step-indexed.

We call these relations heterogeneous because they are parameterized by two types, one for each of the arguments. This property is important for our coherence proof, since it makes it possible to relate the results of the translation of two typing derivations which assign different types to the same term. When both types τ_1 and τ_2 are Nat or both are arrow types, the value relation $\mathcal{V}[\![\tau_1;\tau_2]\!]$ is standard. Two values are related for type Nat if they are the same constant, and two functions are related when they map related arguments to related results. The most interesting are the cases when type parameters of the relation are different. When one of these types is Unit, then any values are in the relation, because we do not expect them to carry any information – Unit is the result of translating the Top type. Functions are never related with natural numbers.

The relation $\mathcal{E}[\![\tau_1;\tau_2]\!]$ for closed terms is defined by biorthogonality. Two terms are related if they behave the same in related contexts, and contexts are related (relation $\mathcal{K}[\![\tau_1;\tau_2]\!]$) if they yield the same observations when plugged with related values. Yielding the same observations (relation $\precsim$) is defined for each step-index separately: $e_1 \precsim e_2$ is valid at k iff termination of e_1 using at most k β-steps (and any number of ι-steps) implies termination of e_2 in any number of steps. This interpretation is monotone, so the relation $\precsim$ can be added to the LSLR logic. The relation $\mathcal{E}[\![\tau_1;\tau_2]\!]$ is extended to open terms as usual: two open

$$
\begin{array}{rcl}
(v_1, v_2) \in \mathcal{V}[\![\mathsf{Nat}; \mathsf{Nat}]\!] & \iff & \exists n, v_1 = v_2 = n \\
(v_1, v_2) \in \mathcal{V}[\![\tau_1' \to \tau_1; \tau_2' \to \tau_2]\!] & \iff & \forall (a_1, a_2) \in \mathcal{V}[\![\tau_1'; \tau_2']\!].(v_1\ a_1, v_2\ a_2) \in \mathcal{E}[\![\tau_1; \tau_2]\!] \\
(v_1, v_2) \in \mathcal{V}[\![\tau_1; \tau_2]\!] & \iff & \top \qquad \text{if } \tau_1 = \mathsf{Unit} \text{ or } \tau_2 = \mathsf{Unit} \\
(v_1, v_2) \in \mathcal{V}[\![\tau_1; \tau_2]\!] & \iff & \bot \qquad \text{otherwise} \\
(e_1, e_2) \in \mathcal{E}[\![\tau_1; \tau_2]\!] & \iff & \forall (E_1, E_2) \in \mathcal{K}[\![\tau_1; \tau_2]\!].E_1[e_1] \precsim E_2[e_2] \\
(E_1, E_2) \in \mathcal{K}[\![\tau_1; \tau_2]\!] & \iff & \forall (v_1, v_2) \in \mathcal{V}[\![\tau_1; \tau_2]\!].E_1[v_1] \precsim E_2[v_2] \\
k \models e_1 \precsim e_2 & \iff & e_1{\downarrow^k} \implies e_2{\downarrow} \\
(\gamma_1, \gamma_2) \in \mathcal{G}[\![\Gamma_1; \Gamma_2]\!] & \iff & \forall x, (\gamma_1(x), \gamma_2(x)) \in \mathcal{V}[\![\Gamma_1(x); \Gamma_2(x)]\!] \\
\Gamma_1; \Gamma_2 \vdash e_1 \precsim_{log} e_2 \ : \ \tau_1; \tau_2 & \iff & \forall (\gamma_1, \gamma_2) \in \mathcal{G}[\![\Gamma_1; \Gamma_2]\!].(e_1\gamma_1, e_2\gamma_2) \in \mathcal{E}[\![\tau_1; \tau_2]\!]
\end{array}
$$

Figure 4 Logical relations for the λ-calculus with explicit coercions

terms are related (written $\Gamma_1; \Gamma_2 \vdash e_1 \precsim_{log} e_2 \ : \ \tau_1; \tau_2$) when every pair of related closing substitutions makes them related.

Notice that we do not assume that related terms have valid types. Our relations may include some "garbage", e.g., $(1, \lambda x.x) \in \mathcal{V}[\![\mathsf{Unit}; \mathsf{Nat}]\!]$, but it is non-problematic. One can mechanically prune these relations to well-typed terms, but this change complicates formalization and we did not find it useful.

In this presentation we consider languages with only one base type. Adding more base types and some subtyping between them will not change the general shape of the proof, but defining logical relations for such a case is a little trickier. We would stipulate that two values v_1 and v_2 are related for base types b_1 and b_2 iff for every common supertype b of b_1 and b_2, coercing v_1 and v_2 to b yields the same constant.

The relation $\precsim$ is preserved by reductions in the following sense, where the third assertion expresses an elimination rule of the later modality that is crucial in the subsequent proofs.

▶ **Lemma 2.** *The following assertions hold:*

1. *If* $e_1 \to_\iota e_1'$ *and* $e_1' \precsim e_2$ *then* $e_1 \precsim e_2$.
2. *If* $e_2 \to_\iota e_2'$ *and* $e_1 \precsim e_2'$ *then* $e_1 \precsim e_2$.
3. *If* $e_1 \to_\beta e_1'$ *and* $\triangleright e_1' \precsim e_2$ *then* $e_1 \precsim e_2$.
4. *If* $e_2 \to_\beta e_2'$ *and* $e_1 \precsim e_2'$ *then* $e_1 \precsim e_2$.

The proof of soundness of the logical relations follows closely the standard technique for biorthogonal logical relations [20, 13]. First, we need to show compatibility lemmas, which state that the relation is preserved by every language construct. Most of them are standard and we omit them due to lack of space. The only compatibility lemma specific to our relations is the following lemma for coercion application.

▶ **Lemma 3** (Adequacy). *If* $(e_1, e_2) \in \mathcal{E}[\![\tau; \tau]\!]$ *then* $e_1 \precsim e_2$.

Proof. Let us show $\square[e_1] \precsim \square[e_2]$. Using the assertion $(e_1, e_2) \in \mathcal{E}[\![\tau; \tau]\!]$, it suffices to show $(\square, \square) \in \mathcal{K}[\![\tau; \tau]\!]$, which is trivial, since values always terminate. ◀

▶ **Lemma 4** (Coercion compatibility). *The logical relation is preserved by coercion application.*

1. *If* $c :: \tau_1 \triangleright \tau_2$ *and* $\Gamma_1; \Gamma_2 \vdash e_1 \precsim_{log} e_2 \ : \ \tau_1; \tau_0$ *then* $\Gamma_1; \Gamma_2 \vdash c\, e_1 \precsim_{log} e_2 \ : \ \tau_2; \tau_0$.
2. *If* $c :: \tau_1 \triangleright \tau_2$ *and* $\Gamma_1; \Gamma_2 \vdash e_1 \precsim_{log} e_2 \ : \ \tau_0; \tau_1$ *then* $\Gamma_1; \Gamma_2 \vdash e_1 \precsim_{log} c\, e_2 \ : \ \tau_0; \tau_2$.

Proof. We prove both cases by induction on the typing derivation of the coercion c. ◀

▶ **Theorem 5** (Fundamental property). *If* $\Gamma \vdash e \ : \ \tau$ *then* $\Gamma; \Gamma \vdash e \precsim_{log} e \ : \ \tau; \tau$.

Proof. By induction on the derivation $\Gamma \vdash e \ : \ \tau$. In each case we apply the corresponding compatibility lemma. ◀

▶ **Lemma 6** (Precongruence). *If* $\vdash C \ : \ (\Gamma;\tau) \ \rightsquigarrow \ \tau_0$ *and* $\Gamma;\Gamma \vdash e_1 \precsim_{log} e_2 \ : \ \tau;\tau$ *then* $(C[e_1], C[e_2]) \in \mathcal{E}[\![\tau_0;\tau_0]\!]$.

Proof. By induction on the derivation of context typing, using the appropriate compatibility lemma in each case. For contexts containing subterms we also need the fundamental property. For the empty context we use the empty substitution, since the empty substitutions are in relation $\mathcal{G}[\![\varnothing;\varnothing]\!]$. ◀

▶ **Theorem 7** (Soundness). *If* $k \models \Gamma;\Gamma \vdash e_1 \precsim_{log} e_2 \ : \ \tau;\tau$ *holds for every* k*, then* $\Gamma \vdash e_1 \precsim_{ctx} e_2 \ : \ \tau$.

Proof. Suppose $\vdash C \ : \ (\Gamma;\tau) \ \rightsquigarrow \ \tau_0$ and $C[e_1]{\downarrow}$, we need to show $C[e_2]{\downarrow}$. By Lemma 6 and Lemma 3 we know $k \models C[e_1] \precsim C[e_2]$ for every k. Taking k to be the number of steps in which $C[e_1]$ terminates, we have that $C[e_2]$ also terminates, by the definition of $\precsim$. ◀

3.4 Coherence of the coercion semantics

Having established soundness of the logical relations, we are in a position to prove the main coherence lemma, phrased in terms of the logical relations, and the coherence theorem.

▶ **Lemma 8.** *If* $D_i :: \Gamma_i \vdash e \ : \ \tau_i$ *for* $i = 1, 2$ *are two typing judgments for the same term* e *of the source language, then* $[\![\Gamma_1]\!];[\![\Gamma_2]\!] \vdash \mathcal{T}[\![e]\!]_{D_1} \precsim_{log} \mathcal{T}[\![e]\!]_{D_2} \ : \ [\![\tau_1]\!];[\![\tau_2]\!]$.

Proof. The proof follows by induction on the structure of both derivations D_1 and D_2. At least one of these derivations is decreased in every case. When one of derivations starts with subsumption rule (T-Sub), we apply Lemma 4. The coercion that we get after translation is well-typed by Lemma 1. In other cases we just apply the appropriate compatibility lemma. ◀

▶ **Theorem 9** (Coherence). *If* D_1 *and* D_2 *are derivations of the same typing judgment* $\Gamma \vdash e \ : \ \tau$*, then* $[\![\Gamma]\!] \vdash \mathcal{T}[\![e]\!]_{D_1} \precsim_{ctx} \mathcal{T}[\![e]\!]_{D_2} \ : \ [\![\tau]\!]$.

Proof. Immediately from Lemma 8 and Theorem 7. ◀

Coercion semantics described here translates the source language into the language with explicit coercions. We chose coercions to be a separate syntactic category, because we found it very convenient, especially for proving Lemma 4. However, one can define a coercion semantics which translates subtyping proofs directly to λ-expressions. Our result can be easily extended for such a translation. Let $|e|$ be a term e with all the coercions replaced by the corresponding expressions. To prove that for any contextually equivalent terms e_1 and e_2 in the language with coercions, terms $|e_1|$ and $|e_2|$ are contextually equivalent in the language without coercions, we need three simple facts that can be easily verified:

1. every well-typed term in the language without coercions is well typed in the language with coercions,
2. term e terminates iff $|e|$ terminates,
3. if context C does not contain coercions then $C[|e|] = |C[e]|$.

In the next section we show that the results presented in this section can be adapted to a considerably more complex calculus – a calculus of delimited control with control-effect subtyping.

$$
\begin{array}{lcll}
\tau & ::= & \mathsf{Nat} \mid \tau \to T & \text{(pure types)} \\
T & ::= & \tau \mid \tau[T]T & \text{(types)} \\
e & ::= & x \mid \lambda x.e \mid e\ e \mid \mathsf{fix}\ x(x).e \mid \mathcal{S}_0 x.e \mid \langle e \rangle \mid n & \text{(expressions)}
\end{array}
$$

$$\frac{}{T \le T}\ \text{S-Refl} \qquad \frac{T_2 \le T_3 \qquad T_1 \le T_2}{T_1 \le T_3}\ \text{S-Trans} \qquad \frac{\tau_1 \le \tau_2 \qquad T_2 \le T_1 \qquad U_1 \le U_2}{\tau_1[T_1]U_1 \le \tau_2[T_2]U_2}\ \text{S-Cons}$$

$$\frac{T_1 \le T_2}{\tau \le \tau[T_1]T_2}\ \text{S-Lift} \qquad \frac{\tau_2 \le \tau_1 \qquad T_1 \le T_2}{(\tau_1 \to T_1) \le (\tau_2 \to T_2)}\ \text{S-Arr} \qquad \frac{\Gamma \vdash e\ :\ T \qquad T \le U}{\Gamma \vdash e\ :\ U}\ \text{T-Sub}$$

$$\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x\ :\ \tau}\ \text{T-Var} \qquad \frac{\Gamma, x : \tau \vdash e\ :\ T}{\Gamma \vdash \lambda x.e\ :\ \tau \to T}\ \text{T-Abs} \qquad \frac{\Gamma \vdash e_1\ :\ \tau \to T \qquad \Gamma \vdash e_2\ :\ \tau}{\Gamma \vdash e_1\ e_2\ :\ T}\ \text{T-PApp}$$

$$\frac{\Gamma \vdash e_1\ :\ (\tau_2 \to \tau_1[U_4]U_3)[U_2]U_1 \qquad \Gamma \vdash e_2\ :\ \tau_2[U_3]U_2}{\Gamma \vdash e_1\ e_2\ :\ \tau_1[U_4]U_1}\ \text{T-App} \qquad \frac{}{\Gamma \vdash n\ :\ \mathsf{Nat}}\ \text{T-Const}$$

$$\frac{\Gamma, f : \tau \to T, x : \tau \vdash e\ :\ T}{\Gamma \vdash \mathsf{fix}\ f(x).e\ :\ \tau \to T}\ \text{T-Fix} \qquad \frac{\Gamma, x : \tau \to T \vdash e\ :\ U}{\Gamma \vdash \mathcal{S}_0 x.e\ :\ \tau[T]U}\ \text{T-Sft} \qquad \frac{\Gamma \vdash e\ :\ \tau[\tau]T}{\Gamma \vdash \langle e \rangle\ :\ T}\ \text{T-Rst}$$

Figure 5 The source language – λ-calculus with delimited control and effect subtyping

4 Coherence of a CPS translation of control-effect subtyping

The calculus of delimited control studied in this section is the λ-calculus extended with recursion and the control operators shift_0 ($\mathcal{S}_0$) and reset_0 ($\langle\cdot\rangle$) [11] that can explore and reorganize an arbitrary portion of the stack of delimited continuations. It was built around a central idea that types for such a calculus should contain information about the stack and that the amount of that information should be governed by a subtyping mechanism [16]. We will define the semantics of the calculus by a CPS translation to a target calculus endowed with a reduction semantics, but if we were to directly give a reduction rule for shift_0, it would be:

$$F[\langle E[\mathcal{S}_0 x.e] \rangle] \quad \to \quad F[e\{\lambda y.\langle E[y] \rangle / x\}]$$

where E is a pure evaluation context representing the current delimited continuation (delimited by $\langle\cdot\rangle$ and captured by $\mathcal{S}_0$) and F is a metacontext, i.e., a list of such pure evaluation contexts separated by control delimiters, representing the current metacontinuation [6]. In terms of abstract-machine semantics E together with F represent the entire control stack. We can see from the reduction rule that nothing prevents the expression e from capturing another delimited continuation from F and, therefore, the types of the calculus express requirements on the shape of the control stack, so that expressions can safely perform control operations exploring the stack. However, under some conditions, an expression that imposes certain requirements on the stack can be used with a stack of which more is known or assumed. Such coercions are possible thanks to the subtyping relation that lies at the heart of the calculus.

4.1 The lambda calculus with delimited control and effect subtyping

The syntax and typing rules of the calculus of delimited control are shown in Figure 5. Types are either pure (τ) or effect annotated ($\tau[T_1]T_2$). An expression of type $\tau[T_1]T_2$ can be used with the top-most delimited continuation that when given a value of type τ behaves as specified by its answer type T_1, and with the rest of the stack whose type is T_2.

The calculus comprises the simply typed lambda calculus with the standard subtyping rules, extended with typing and subtyping rules for effectful computations. The most

$$
\begin{array}{rcll}
\tau & ::= & \mathsf{Nat} \mid \tau \to T & \text{(pure types)} \\
T & ::= & \tau \mid (\tau \to T) \Rightarrow T & \text{(types)} \\
c & ::= & \mathsf{id} \mid c \circ c \mid c \to c \mid \uparrow c \mid c[c]c & \text{(coercions)} \\
e & ::= & x \mid \lambda x.e \mid e\ e \mid c\ e \mid \mathsf{fix}\ x(x).e \mid n & \text{(expressions)} \\
v & ::= & x \mid \lambda x.e \mid \mathsf{fix}\ x(x).e \mid (c \to c)\ v \mid \uparrow c\ v \mid (c[c]c)\ v \mid n & \text{(values)} \\
E & ::= & \Box \mid E\ e \mid v\ E \mid c\ E & \text{(evaluation contexts)}
\end{array}
$$

$$\frac{}{\mathsf{id} :: T \triangleright T}\ \text{S-Refl} \qquad \frac{c_1 :: T_2 \triangleright T_3 \qquad c_2 :: T_1 \triangleright T_2}{c_1 \circ c_2 :: T_1 \triangleright T_3}\ \text{S-Trans}$$

$$\frac{c_1 :: \tau_2 \triangleright \tau_1 \qquad c_2 :: T_1 \triangleright T_2}{c_1 \to c_2 :: (\tau_1 \to T_1) \triangleright (\tau_2 \to T_2)}\ \text{S-Arr}$$

$$\frac{c :: T_1 \triangleright T_2}{\uparrow c :: \tau \triangleright ((\tau \to T_1) \Rightarrow T_2)}\ \text{S-Lift} \qquad \frac{c :: \tau_1 \triangleright \tau_2 \qquad c_1 :: T_2 \triangleright T_1 \qquad c_2 :: U_1 \triangleright U_2}{c[c_1]c_2 :: ((\tau_1 \to T_1) \Rightarrow U_1) \triangleright ((\tau_2 \to T_2) \Rightarrow U_2)}\ \text{S-Cons}$$

$$\frac{(x : \tau) \in \Gamma}{\Gamma \vdash x\ :\ \tau}\ \text{T-Var} \qquad \frac{\Gamma, x : \tau \vdash e\ :\ T}{\Gamma \vdash \lambda x.e\ :\ \tau \to T}\ \text{T-Abs} \qquad \frac{\Gamma \vdash e_1\ :\ \tau \to T \qquad \Gamma \vdash e_2\ :\ \tau}{\Gamma \vdash e_1\ e_2\ :\ T}\ \text{T-App}$$

$$\frac{\Gamma, x : \tau \to T \vdash e\ :\ U}{\Gamma \vdash \lambda x.e\ :\ (\tau \to T) \Rightarrow U}\ \text{T-KAbs} \qquad \frac{\Gamma \vdash e\ :\ (\tau \to T) \Rightarrow U \qquad \Gamma \vdash v\ :\ \tau \to T}{\Gamma \vdash e\ v\ :\ U}\ \text{T-KApp}$$

$$\frac{c :: T \triangleright U \qquad \Gamma \vdash e\ :\ T}{\Gamma \vdash c\ e\ :\ U}\ \text{T-CApp} \qquad \frac{\Gamma, f : \tau \to T, x : \tau \vdash e\ :\ T}{\Gamma \vdash \mathsf{fix}\ f(x).e\ :\ \tau \to T}\ \text{T-Fix} \qquad \frac{}{\Gamma \vdash n\ :\ \mathsf{Nat}}\ \text{T-Const}$$

$$
\begin{array}{rcl}
E[(\lambda x.e)\ v] & \to_\beta & E[e\{v/x\}] \\
E[(\mathsf{fix}\ f(x).e)\ v] & \to_\beta & E[e\{\mathsf{fix}\ f(x).e/f, v/x\}] \\
E[\uparrow c\ v_1\ v_2] & \to_\beta & E[c\ (v_2\ v_1)]
\end{array}
\qquad
\begin{array}{rcl}
E[\mathsf{id}\ v] & \to_\iota & E[v] \\
E[(c_1 \circ c_2)\ v] & \to_\iota & E[c_1\ (c_2\ v)] \\
E[(c_1 \to c_2)\ v_1\ v_2] & \to_\iota & E[c_2\ (v_1\ (c_1\ v_2))] \\
E[(c[c_1]c_2)\ v_1\ v_2] & \to_\iota & E[c_2\ (v_1\ ((c \to c_1)\ v_2))]
\end{array}
$$

Figure 6 The target language – λ-calculus with explicit coercions of control effects

interesting from the point of view of effect subtyping are the rules S-Cons and S-Lift. The rule S-Cons is a direct consequence of the interpretation of effect-annotated types given above that actually follows from the CPS interpretation of delimited continuations – a type $\tau[T_1]T_2$ is interpreted in CPS as $(\tau \to T_1) \Rightarrow T_2$, where $\Rightarrow$ means an effectful function space (see Section 4.2). The rule S-Lift is more interesting and it says that a pure computation can be considered impure, provided the answer type of the top-most delimited continuation can be coerced into the type of the rest of the stack. We have, for an example, $\mathsf{Nat} \le \mathsf{Nat}[\mathsf{Nat}]\mathsf{Nat}$. It is worth noting that the calculus provides two rules for function application. They are necessary for expressivity reasons, but at the same time they are an additional source of difficulty when it comes to establishing the coherence of the subtyping. For more detailed presentation of the calculus we refer the reader to [16]. (The presentation of the type system in [16] differs from the one shown in Figure 5 in some inessential details, but the two type systems are equally expressive.)

4.2 Coercion semantics: a type-directed selective CPS translation

The syntax and typing rules of the target language are presented in Figure 6. There are two kinds of arrow type: the usual one $\tau \to T$ for regular functions and the effectful one $(\tau \to T) \Rightarrow U$ for expressions in CPS. We make this distinction to express the fact that the CPS translation (see Figure 7) yields expressions with strong restrictions on the occurrence of terms in CPS: they are never passed as arguments (typing environment consists of only

$$\begin{array}{rclcrcl} [\![\mathsf{Nat}]\!]_p & = & \mathsf{Nat} & & [\![\tau]\!] & = & [\![\tau]\!]_p \\ [\![\tau \to T]\!]_p & = & [\![\tau]\!]_p \to [\![T]\!] & & [\![\tau[T]U]\!] & = & ([\![\tau]\!]_p \to [\![T]\!]) \Rightarrow [\![U]\!] \end{array}$$

$$\begin{array}{rcl} \mathcal{S}[\![T \le T]\!]_{\text{S-Refl}} & = & \mathsf{id} \\ \mathcal{S}[\![T_1 \le T_3]\!]_{\text{S-Trans}(D_1,D_2)} & = & \mathcal{S}[\![T_2 \le T_3]\!]_{D_1} \circ \mathcal{S}[\![T_1 \le T_2]\!]_{D_2} \\ \mathcal{S}[\![\tau_1 \to T_1 \le \tau_2 \to T_2]\!]_{\text{S-Arr}(D_1,D_2)} & = & \mathcal{S}[\![\tau_2 \le \tau_1]\!]_{D_1} \to \mathcal{S}[\![T_1 \le T_2]\!]_{D_2} \\ \mathcal{S}[\![\tau \le \tau[T]U]\!]_{\text{S-Lift}(D)} & = & \uparrow \mathcal{S}[\![T \le U]\!]_D \\ \mathcal{S}[\![\tau_1[T_1]U_1 \le \tau_2[T_2]U_2]\!]_{\text{S-Cons}(D,D_1,D_2)} & = & \mathcal{S}[\![\tau_1 \le \tau_2]\!]_D[\mathcal{S}[\![T_2 \le T_1]\!]_{D_1}]\mathcal{S}[\![U_1 \le U_2]\!]_{D_2} \end{array}$$

$$\begin{array}{rcl@{\qquad}rcl} \mathcal{T}[\![x]\!]_{\text{T-Var}} & = & x & \mathcal{T}[\![\mathcal{S}_0 x.e]\!]_{\text{T-Sft}(D)} & = & \lambda x.\mathcal{T}[\![e]\!]_D \\ \mathcal{T}[\![\lambda x.e]\!]_{\text{T-Abs}(D)} & = & \lambda x.\mathcal{T}[\![e]\!]_D & \mathcal{T}[\![\langle e \rangle]\!]_{\text{T-Rst}(D)} & = & \mathcal{T}[\![e]\!]_D\ (\lambda x.x) \\ \mathcal{T}[\![\mathsf{fix}\, f(x).e]\!]_{\text{T-Fix}(D)} & = & \mathsf{fix}\, f(x).\mathcal{T}[\![e]\!]_D & \mathcal{T}[\![n]\!]_{\text{T-Const}} & = & n \\ \mathcal{T}[\![e]\!]_{\text{T-Sub}(D_1,D_2)} & = & \mathcal{S}[\![T \le U]\!]_{D_2}\ \mathcal{T}[\![e]\!]_{D_1} & & & \\ \mathcal{T}[\![e_1\ e_2]\!]_{\text{T-PApp}(D_1,D_2)} & = & \mathcal{T}[\![e_1]\!]_{D_1}\ \mathcal{T}[\![e_2]\!]_{D_2} & & & \\ \mathcal{T}[\![e_1\ e_2]\!]_{\text{T-App}(D_1,D_2)} & = & \lambda k.\mathcal{T}[\![e_1]\!]_{D_1}\ (\lambda f.\mathcal{T}[\![e_2]\!]_{D_2}\ (\lambda x.f\ x\ k)) & & & \end{array}$$

Figure 7 Type-directed selective CPS translation

pure types) and they can be applied only to values (witness the rule T-KApp) representing continuations.

Again, the operational semantics distinguishes between β-rules and ι-rules. We classified the last β-rule as "actual computation" because it does not only rearrange coercions. It translates back a lifted value v_1 and applies to it a given continuation v_2. This rule and the last ι-rule reduce a coerced value applied to a continuation, so terms of the form $(\uparrow c\, v)$ and $(c[c]c\, v)$ are considered values. Notice that these values have effectful types. We extend the notion of ι-reduction to evaluation contexts: $E_1 \to_\iota E_2$ holds iff $E_1[v] \to_\iota E_2[v]$ for every value v.

As in Section 3, the metavariable C denotes general closed contexts. We also define typing of general contexts $\vdash C\ :\ (\Gamma;T) \rightsquigarrow T_0$ as before. The definition of contextual approximation $\Gamma \vdash e_1 \precsim_{ctx} e_2\ :\ T$ is slightly weaker, because we allow only contexts with pure answer type. Indeed, an expression that requires a continuation to trigger computation can hardly be considered a complete program.

The coercion semantics of the source language is given by the type-directed selective CPS translation presented in Figure 7. The translation is selective because it leaves terms of pure type in direct style – witness, e.g, the equations for variable or pure application. Effectful applications are translated according to Plotkin's call-by-value CPS translation [21], whereas the translation of shift_0 and reset_0 is surprisingly straightforward – shift_0 is turned into a lambda-abstraction expecting a delimited continuation, and reset_0 is interpreted by providing its subexpression with the reset delimited continuation, represented by the identity function.

▶ **Example 10.** Consider the program $(\mathsf{fix}\, f(x).f\ x)\ 1$ in the source language. We derive the type $\mathsf{Nat}[T]T$ for it in two ways: let D_1 be the derivation

$$\dfrac{\dfrac{\dfrac{\vdots}{f : \mathsf{Nat} \to \mathsf{Nat}[T]T, x : \mathsf{Nat} \vdash f\ x\ :\ \mathsf{Nat}[T]T}}{\vdash \mathsf{fix}\, f(x).f\ x\ :\ \mathsf{Nat} \to \mathsf{Nat}[T]T}\ \text{T-Fix} \qquad \dfrac{}{\vdash 1\ :\ \mathsf{Nat}}\ \text{T-Const}}{\vdash (\mathsf{fix}\, f(x).f\ x)\ 1\ :\ \mathsf{Nat}[T]T}\ \text{T-PApp}$$

$$
\begin{array}{rcl}
(v_1, v_2) \in \mathcal{V}[\![\mathsf{Nat}; \mathsf{Nat}]\!] & \iff & \exists n, v_1 = v_2 = n \\
(v_1, v_2) \in \mathcal{V}[\![\tau_1 \to T_1; \tau_2 \to T_2]\!] & \iff & \forall (a_1, a_2) \in \mathcal{V}[\![\tau_1; \tau_2]\!].(v_1\ a_1, v_2, a_2\ ) \in \mathcal{E}[\![T_1; T_2]\!] \\
(v_1, v_2) \in \mathcal{V}[\![\tau_1; \tau_2]\!] & \iff & \bot \qquad \text{otherwise} \\
(e_1, e_2) \in \mathcal{E}[\![T_1; T_2]\!] & \iff & \forall (E_1, E_2) \in \mathcal{K}[\![T_1; T_2]\!].E_1[e_1] \precsim E_2[e_2] \\
(E_1, E_2) \in \mathcal{K}[\![\tau_1; \tau_2]\!] & \iff & \forall (v_1, v_2) \in \mathcal{V}[\![\tau_1; \tau_2]\!].E_1[v_1] \precsim E_2[v_2] \\
(E_1, E_2) \in \mathcal{K}[\![\tau_1; (\tau_2 \to T_2) \Rightarrow U_2]\!] & \iff & \exists T, (E, k) \in \mathcal{KV}[\![\tau_1 \rightsquigarrow T; \tau_2 \to T_2]\!], \\
 & & \quad (E_1', E_2') \in \mathcal{K}[\![T; U_2]\!]. \\
 & & \quad\quad E_1 \to_\iota^* E_1'[E[\,]] \wedge E_2 \to_\iota^* E_2'[[\,]\ k] \\
(E_1, E_2) \in \mathcal{K}[\![(\tau_1 \to T_1) \Rightarrow U_1; \tau_2]\!] & \iff & \exists T, (k, E) \in \rhd\mathcal{VK}[\![\tau_1 \to T_1; \tau_2 \rightsquigarrow T]\!], \\
 & & \quad (E_1', E_2') \in \rhd\mathcal{K}[\![U_1; T]\!]. \\
 & & \quad\quad E_1 \to_\iota^* E_1'[[\,]\ k] \wedge E_2 \to_\iota^* E_2'[E[\,]] \\
(E_1, E_2) \in \mathcal{K}[\![(\tau_1 \to T_1) \Rightarrow U_1; & & \\
\quad (\tau_2 \to T_2) \Rightarrow U_2]\!] & \iff & \exists (k_1, k_2) \in \mathcal{V}[\![\tau_1 \to T_1; \tau_2 \to T_2]\!], \\
 & & \quad (E_1', E_2') \in \mathcal{K}[\![U_1; U_2]\!]. \\
 & & \quad\quad E_1 \to_\iota^* E_1'[[\,]\ k_1] \wedge E_2 \to_\iota^* E_2'[[\,]\ k_2] \\
(E, v) \in \mathcal{KV}[\![\tau_1 \rightsquigarrow T_1; \tau_2 \to T_2]\!] & \iff & \forall (a_1, a_2) \in \mathcal{V}[\![\tau_1; \tau_2]\!].(E[a_1], v\ a_2) \in \mathcal{E}[\![T_1; T_2]\!] \\
(v, E) \in \mathcal{VK}[\![\tau_1 \to T_1; \tau_2 \rightsquigarrow T_2]\!] & \iff & \forall (a_1, a_2) \in \mathcal{V}[\![\tau_1; \tau_2]\!].(v\ a_1, E[a_2]) \in \mathcal{E}[\![T_1; T_2]\!] \\
(\gamma_1, \gamma_2) \in \mathcal{G}[\![\Gamma_1; \Gamma_2]\!] & \iff & \forall x.(\gamma_1(x), \gamma_2(x)) \in \mathcal{V}[\![\Gamma_1(x); \Gamma_2(x)]\!] \\
\Gamma_1; \Gamma_2 \vdash e_1 \precsim_{log} e_2 \ :\ T_1; T_2 & \iff & \forall (\gamma_1, \gamma_2) \in \mathcal{G}[\![\Gamma_1; \Gamma_2]\!].(e_1\gamma_1, e_2\gamma_2) \in \mathcal{E}[\![T_1; T_2]\!]
\end{array}
$$

Figure 8 Logical relations for the λ-calculus with explicit coercions of control effects

and D_2 be the derivation

$$
\dfrac{\dfrac{\dfrac{\dots}{\vdash \mathsf{fix}\ f(x).f\ x\ :\ \mathsf{Nat} \to \mathsf{Nat}[T]T} \quad \dots}{\vdash \mathsf{fix}\ f(x).f\ x\ :\ (\mathsf{Nat} \to \mathsf{Nat}[T]T)[T]T}\ \text{T-Sub} \qquad \dfrac{\dfrac{}{\vdash 1\ :\ \mathsf{Nat}}\ \text{T-Const} \quad \dots}{\vdash 1\ :\ \mathsf{Nat}[T]T}\ \text{T-Sub}}{\vdash (\mathsf{fix}\ f(x).f\ x)\ 1\ :\ \mathsf{Nat}[T]T}\ \text{T-App}
$$

Then

$$
\begin{array}{rcl}
\mathcal{T}[\![(\mathsf{fix}\ f(x).f\ x)\ 1]\!]_{D_1} & = & (\mathsf{fix}\ f(x).f\ x)\ 1 \\
\mathcal{T}[\![(\mathsf{fix}\ f(x).f\ x)\ 1]\!]_{D_2} & = & \lambda k.(\uparrow \mathsf{id}\ (\mathsf{fix}\ f(x).f\ x))\ (\lambda g.(\uparrow \mathsf{id}\ 1)\ (\lambda y.g\ y\ k))
\end{array}
$$

By the results of the next sections, these terms are contextually equivalent.

▶ **Lemma 11.** *Coercion semantics preserves types.*

1. *If* $D :: T_1 \le T_2$ *then* $\mathcal{S}[\![T_1 \le T_2]\!]_D :: [\![T_1]\!] \rhd [\![T_2]\!]$.
2. *If* $D :: \Gamma \vdash e\ :\ T$ *then* $[\![\Gamma]\!] \vdash \mathcal{T}[\![e]\!]_D\ :\ [\![T]\!]$.

4.3 Logical relations

The logical relations are defined in Figure 8. The relation $\mathcal{V}[\![\tau_1; \tau_2]\!]$ for pure values and the relation $\mathcal{E}[\![T_1; T_2]\!]$ for expressions are similar to the relations defined in Section 3.3. All information about control effects is captured in the relation $\mathcal{K}[\![T_1; T_2]\!]$ for contexts. If T_1 and T_2 are computation arrow types, then two contexts are related iff they can be decomposed as applications to related continuations in related contexts. In general, this application to a continuation does not have to be the top-most element of the context and some rearrangement of coercions is needed, so such a decomposition is defined by ι-reduction of contexts.

The most interesting are the cases that relate pure and impure contexts. As previously, the impure context should be decomposed to a continuation k and the rest of the context. Then the pure context should be decomposed in such a way that the continuation k is related with some portion E of the pure context. The answer type of E cannot be retrieved from the type of the initial pure context, so we quantify over all possible types. In order to make the construction well-founded, the relations are defined by nested induction on step indices and on the structure of the second type. Notice that step indices play a role only in one case – when we quantify over the second type and the later operator guards the non-structural use of the relations $\mathcal{VK}[\![\tau_1 \to T_1; \tau_2 \rightsquigarrow T]\!]$ and $\mathcal{K}[\![U_1; T]\!]$. The auxiliary relations $\mathcal{KV}[\![\tau_1 \rightsquigarrow T_1; \tau_2 \to T_2]\!]$ and $\mathcal{VK}[\![\tau_1 \to T_1; \tau_2 \rightsquigarrow T_2]\!]$ relate a portion of an evaluation context with a value of an arrow type and they are defined analogously to the value relation for functions.

The relations of this section possess properties analogous to the ones of Section 3.3, in particular the relation $\precsim$ is preserved by reduction (Lemma 2) and the compatibility lemmas (including Lemma 4) hold. However, the proof of the compatibility lemmas requires the following results that establish the preservation of relations with respect to ι-reductions of evaluation contexts.

▶ **Lemma 12.** *The following assertions hold:*
1. *If* $E \to_\iota^* E'$ *and* $E'[e_1] \precsim e_2$ *then* $E[e_1] \precsim e_2$.
2. *If* $E \to_\iota^* E'$ *and* $e_1 \precsim E'[e_2]$ *then* $e_1 \precsim E[e_2]$.
3. *If* $E_1 \to_\iota^* E_1'$ *and* $(E_1', E_2) \in \mathcal{K}[\![T_1; T_2]\!]$ *then* $(E_1, E_2) \in \mathcal{K}[\![T_1; T_2]\!]$.
4. *If* $E_2 \to_\iota^* E_2'$ *and* $(E_1, E_2') \in \mathcal{K}[\![T_1; T_2]\!]$ *then* $(E_1, E_2) \in \mathcal{K}[\![T_1; T_2]\!]$.

The rest of the soundness proof follows the same lines as in Section 3.3. Interestingly, the adequacy lemma can be proved only for pure types, which is in harmony with the notion of contextual equivalence in the target calculus.

▶ **Theorem 13** (Fundamental property). *If* $\Gamma \vdash e \ : \ T$ *then* $\Gamma; \Gamma \vdash e \precsim_{log} e \ : \ T; T$.

▶ **Lemma 14** (Precongruence). *If* $\vdash C \ : \ (\Gamma; T) \ \rightsquigarrow \ \tau$ *and* $\Gamma; \Gamma \vdash e_1 \precsim_{log} e_2 \ : \ T; T$, *then* $(C[e_1], C[e_2]) \in \mathcal{E}[\![\tau; \tau]\!]$.

▶ **Lemma 15** (Adequacy). *If* $(e_1, e_2) \in \mathcal{E}[\![\tau; \tau]\!]$ *then* $e_1 \precsim e_2$.

▶ **Theorem 16** (Soundness). *If* $k \models \Gamma; \Gamma \vdash e_1 \precsim_{log} e_2 \ : \ T; T$ *holds for every* k, *then* $\Gamma \vdash e_1 \precsim_{ctx} e_2 \ : \ T$.

4.4 Coherence of the CPS translation

Although standard compatibility lemmas and coercion compatibility suffice to prove soundness of logical relations, we need another kind of compatibility to prove coherence, since there is another source of ambiguity. Two typing derivations in the source language can be different not only because of the subsumption rule, but also because of two rules for application.

▶ **Lemma 17** (Mixed application compatibility). *The following assertions hold:*
1. *If* $\Gamma_1; \Gamma_2 \vdash f_1 \precsim_{log} f_2 \ : \ ((\tau_1' \to (\tau_1 \to U_4) \Rightarrow U_3) \to U_2) \Rightarrow U_1; \tau_2' \to T_2$
   and $\Gamma_1; \Gamma_2 \vdash e_1 \precsim_{log} e_2 \ : \ (\tau_1' \to U_3) \Rightarrow U_2; \tau_2'$
   then $\Gamma_1; \Gamma_2 \vdash \lambda k. f_1 \ (\lambda f. e_1 \ (\lambda x. f \ x \ k)) \precsim_{log} f_2 \ e_2 \ : \ (\tau_1' \to U_4) \Rightarrow U_1; T$.
2. *If* $\Gamma_1; \Gamma_2 \vdash f_1 \precsim_{log} f_2 \ : \ \tau_1' \to T_1; ((\tau_2' \to (\tau_2 \to U_4) \Rightarrow U_3) \to U_2) \Rightarrow U_1$
   and $\Gamma_1; \Gamma_2 \vdash e_1 \precsim_{log} e_2 \ : \ \tau_1'; (\tau_2' \to U_3) \Rightarrow U_2$
   then $\Gamma_1; \Gamma_2 \vdash f_1 \ e_1 \precsim_{log} \lambda k. f_2 \ (\lambda f. e_2 \ (\lambda x. f \ x \ k)) \ : \ T; (\tau_2' \to U_4) \Rightarrow U_1$.

Proof. Both cases are similar, so we show only the first one. We have to show that both terms closed by substitutions have the same observations in related contexts $(E_1, E_2) \in \mathcal{K}[\![(\tau_1' \to U_4) \Rightarrow U_1; T]\!]$. Since context E_1 is in relation for effectful type, by the definition of logical relations and Lemma 12, it can be decomposed as a continuation k and the rest of the context. Now we have the missing continuation k that can trigger computation in the first term, so the rest of the proof consists in simple context manipulations, applying definitions and performing reductions. ◀

▶ **Lemma 18.** *If $D_i :: \Gamma_i \vdash e \ : \ T_i$ for $i = 1, 2$ are two typing judgments for the same term e of the source language, then $[\![\Gamma_1]\!]; [\![\Gamma_2]\!] \vdash \mathcal{T}[\![e]\!]_{D_1} \precsim_{log} \mathcal{T}[\![e]\!]_{D_2} \ : \ [\![T_1]\!]; [\![T_2]\!]$.*

▶ **Theorem 19** (Coherence)**.** *If D_1 and D_2 are derivations of the same typing judgment $\Gamma \vdash e \ : \ T$, then $[\![\Gamma]\!] \vdash \mathcal{T}[\![e]\!]_{D_1} \precsim_{ctx} \mathcal{T}[\![e]\!]_{D_2} \ : \ [\![T]\!]$.*

In contrast to the calculus considered in Section 3.4, such coherence theorem does not imply coherence of translation directly to the simply typed λ-calculus (where coercions are expressed as λ-terms). As a counterexample, the derivations from Example 10 give us terms that can be distinguished by the context $C = (\lambda x.1)\,[\,]$. This is because types in the target language carry more information than simple types, and in particular, an expression of a type $(\tau \to T) \Rightarrow U$ is not a usual function, but a computation waiting for a continuation, as explained in Section 4.2.

But still we can prove some interesting properties of a direct translation to the simply typed λ-calculus in two cases: when control effects do not leak to the context or when we relate only whole programs. Let $|e|$ be a term e with all coercions replaced by corresponding expressions.

▶ **Corollary 20.** *If $D_1, D_2 :: \Gamma \vdash e \ : \ \tau$ and τ does not contain any type of the form $\tau'[T]U$, then $|\mathcal{T}[\![e]\!]_{D_1}|$ and $|\mathcal{T}[\![e]\!]_{D_2}|$ are contextually equivalent.*

▶ **Corollary 21.** *If $D_1, D_2 :: \Gamma \vdash e \ : \ \tau$ then $|\mathcal{T}[\![e]\!]_{D_1}|$ terminates iff $|\mathcal{T}[\![e]\!]_{D_2}|$ terminates. Moreover, if $\tau = \mathsf{Nat}$ and one of the expressions terminates to a constant, then the other term evaluates to the same constant.*

5 Conclusion

We have shown that the technique of logical relations can be used for establishing the coherence of subtyping, when it is phrased in terms of contextual equivalence in the target of the coercion translation. In particular, we have demonstrated that a combination of heterogeneity, biorthogonality and step-indexing provides a sufficiently powerful tool for establishing coherence of effect subtyping in a calculus of delimited control with the coercion semantics given by a type-directed selective CPS translation. However, we have successfully applied the presented approach also to other calculi with subtyping, e.g., as demonstrated in this article for the simply-typed λ-calculus with recursion. The Coq development accompanying this paper is based on a new embedding of Dreyer et al.'s logic for reasoning about step-indexing that, we believe, considerably improves the presentation and formalization of the logical relations.

Regarding logical relations for type-and-effect systems, there has been a work on proving correctness of a partial evaluator for shift and reset by Asai [5], and on termination of evaluation of the λ-calculi with delimited-control operators by Biernacka et al. [6, 7] and by Materzok and the first author [16]. Unsurprisingly, all these results, like ours, are built on the notion of biorthogonality, even if not mentioned explicitly. The distinctive feature of our

construction is a combination of heterogeneity and step-indexing that supports reasoning about the observational equivalence of terms of different types whose structure is very distant from each other, e.g., about direct-style and continuation-passing-style terms.

Acknowledgments We thank Andrés A. Aristizábal, Małgorzata Biernacka, and the anonymous reviewers for helpful comments on the presentation of this work.

This work has been supported by the Polish National Science Center, grant no. DEC-011/03/B/ST6/00348.

References

1 Amal Ahmed, Derek Dreyer, and Andreas Rossberg. State-dependent representation independence. In *POPL'09*, Savannah, GA, USA, 2009, pp. 340–353.

2 Amal J. Ahmed. Step-indexed syntactic logical relations for recursive and quantified types. In *ESOP'06*, Vienna, Austria, March 2006, *LNCS* 3924, pp. 69–83.

3 Andrew W. Appel and David McAllester. An indexed model of recursive types for foundational proof-carrying code. *ACM TOPLAS*, 23(5):657–683, 2001.

4 Andrew W. Appel, Paul-André Melliès, Christopher D. Richards, and Jérôme Vouillon. A very modal model of a modern, major, general type system. In *POPL'07*, Nice, France, 2007, pp. 109–122.

5 Kenichi Asai. Logical relations for call-by-value delimited continuations. In *Trends in Functional Programming 2005*, Tallinn, Estonia, 2005, pp. 413–428.

6 Małgorzata Biernacka and Dariusz Biernacki. Context-based proofs of termination for typed delimited-control operators. In *PPDP'09*, Coimbra, Portugal, 2009, pp. 289–300.

7 Małgorzata Biernacka, Dariusz Biernacki, and Sergueï Lenglet. Typing control operators in the CPS hierarchy. In *PPDP'11*, Odense, Denmark, 2011, pp. 149–160.

8 Val Breazu-Tannen, Thierry Coquand, Carl A. Gunter, and Andre Scedrov. Inheritance as implicit coercion. *Information and Computation*, 93(1):172–221, 1991.

9 Pierre-Louis Curien and Giorgio Ghelli. Coherence of subsumption, minimum typing and type-checking in $F_{\leq}$. *Mathematical Structures in Computer Science*, 2(1):55–91, 1992.

10 Olivier Danvy and Andrzej Filinski. A functional abstraction of typed contexts. DIKU Rapport 89/12, DIKU, University of Copenhagen, Copenhagen, Denmark, 1989.

11 Olivier Danvy and Andrzej Filinski. Abstracting control. In *Lisp and Functional Programming 1990*, Nice, France, 1990, pp. 151–160.

12 Derek Dreyer, Amal Ahmed, and Lars Birkedal. Logical step-indexed logical relations. *Logical Methods in Computer Science*, 7(2:16):1–37, 2011.

13 Derek Dreyer, Georg Neis, and Lars Birkedal. The impact of higher-order state and control effects on local relational reasoning. *Journal of Functional Programming*, 22(4-5):477–528, 2012.

14 Jean-Louis Krivine. Classical logic, storage operators and second-order lambda-calculus. *Annals of Pure and Applied Logic*, 68(1):53–78, 1994.

15 Marek Materzok. Axiomatizing subtyped delimited continuations. In *CSL'13*, Torino, Italy, 2013, *LIPIcs* 23, pp. 521–539.

16 Marek Materzok and Dariusz Biernacki. Subtyping delimited continuations. In *ICFP'11*, Tokyo, Japan, 2011, pp. 81–93.

17 Marek Materzok and Dariusz Biernacki. A dynamic interpretation of the CPS hierarchy. In *APLAS'12*, LNCS 7705, Kyoto, Japan, 2012, pp. 296–311.

18 John C. Mitchell. *Foundations for Programming Languages*. MIT Press, 1996.

19 James H. Morris. *Lambda Calculus Models of Programming Languages*. PhD thesis, Massachusetts Institute of Technology, 1968.

20 Andrew Pitts and Ian Stark. Operational reasoning for functions with local state. In *Higher Order Operational Techniques in Semantics*, pp. 227–273. 1998.

21 Gordon D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975.

22 John C. Reynolds. The coherence of languages with intersection types. In *Theoretical Aspects of Computer Software 1991*, Sendai, Japan, 1991, *LNCS* 526, pp. 675–700.

23 Tiark Rompf, Ingo Maier, and Martin Odersky. Implementing first-class polymorphic delimited continuations by a type-directed selective CPS-transform. In *ICFP'09*, Edinburgh, UK, 2009, pp. 317–328.

24 Jan Schwinghammer. Coherence of subsumption for monadic types. *Journal of Functional Programming*, 19(2):157–172, 2009.